

**PATENT**  
**5760-16000**  
**VRTS0523**

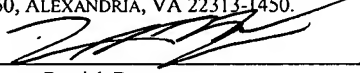
"EXPRESS MAIL" MAILING LABEL NUMBER

EV318248105US

DATE OF DEPOSIT

11-26-03

I HEREBY CERTIFY THAT THIS PAPER OR  
FEE IS BEING DEPOSITED WITH THE  
UNITED STATES POSTAL SERVICE  
"EXPRESS MAIL POST OFFICE TO  
ADDRESSEE" SERVICE UNDER 37 C.F.R. 1.10  
ON THE DATE INDICATED ABOVE AND IS  
ADDRESSED TO THE COMMISSIONER FOR  
PATENTS, BOX PATENT APPLICATION, P.O.  
Box 1450, ALEXANDRIA, VA 22313-1450.

  
Derrick Brown

SYSTEM AND METHOD FOR DETECTING AND STORING FILE CONTENT ACCESS  
INFORMATION WITHIN A FILE SYSTEM

By:

Dhrubajyoti Borthakur

Nur Premo

Joseph Pasqua

Atty. Dkt. No.: 5760-16000/VRTS0523

B. Noël Kivlin/AMP  
Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.  
P.O. Box 398  
Austin, TX 78767-0398  
Ph: (512) 853-8800

## BACKGROUND OF THE INVENTION

### Field of the Invention

5    **[0001]**    This invention relates to computer systems and, more particularly, to storage systems.

### Description of the Related Art

10   **[0002]**    Computer systems often process large quantities of information, including application data and executable code configured to process such data. In numerous embodiments, computer systems provide various types of mass storage devices configured to store data, such as magnetic and optical disk drives, tape drives, etc. To provide a regular and systematic interface through which to access their stored data, such  
15   storage devices are frequently organized into hierarchies of files by software such as an operating system. Often a file defines a minimum level of data granularity that a user can manipulate within a storage device, although various applications and operating system processes may operate on data within a file at a lower level of granularity than the entire file.

20   **[0003]**    In many conventional file-based computer systems, files may be created, destroyed and manipulated with relatively few constraints. Typically, files may be arbitrarily named, subject to operating system conventions, and often, unlimited numbers of exact copies of existing files may be made with ease, subject only to available storage  
25   capacity. While such ease of data proliferation may simplify system operation for the user, it may also result in inefficient use of storage devices. For example, storage devoted to multiple identical copies of a given file may be redundant and therefore wasted. Further, if a user creates multiple copies of a given file, gives each a unique

identity, and then proceeds to work with each file individually, the relationships among files (such as their common origin, type, and degree of common content) may be obscured over time. Still further, not all types of files may be equally well suited to a given type of storage available in a system. For example, recently used data files may be more likely to be used again in the future and therefore good candidates to be stored in faster storage such as a disk drive, but files unlikely to be used again may be better suited to be stored on a tape drive.

[0004] Attempting to track file operations as they occur, to thereby gather greater information about such operations, is complicated by the problem of how such operations may be detected. In most operating system embodiments, application programs may be isolated from one another during execution such that one application may only detect the effects of another, such as a write to a given file, after the fact. However, at the point a file operation (e.g., a modification or copy operation) is visible to another application, the operation may have already occurred and information regarding the source of the operation may no longer be available.

## SUMMARY OF THE INVENTION

[0005] Various embodiments of a system and method for detecting and storing file content access information within a file system are disclosed. In one embodiment, the system may include a storage device configured to store a plurality of files and a file system configured to manage access to the storage device. The file system may be configured to detect an operation to access content of a first file stored on the storage device and, subsequent to detecting the operation, store a record of the operation associated with the first file, where the record includes a signature corresponding to the first file.

[0006] In one specific implementation of the system, the operation may correspond to a file read operation, or a file write operation. In another specific implementation of the system, the operation may be generated by an application and the record may include an identity of the application. In yet another specific implementation of the system, the record may be stored in a named stream corresponding to the first file, the file system may include a history stream, and wherein the file system may be further configured to store an indication of the operation in the history stream in response to storing the record in the named stream.

[0007] A method is also contemplated which, in one embodiment, may include storing a plurality of files, detecting an operation to access content of a first stored file, and subsequent to detecting the operation, storing a record of the operation associated with the first stored file, wherein the record includes a signature corresponding to the first stored file.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

**[0008]** FIG. 1 is a block diagram illustrating one embodiment of a storage system.

5 **[0009]** FIG. 2 is a block diagram illustrating one embodiment of an operating system architecture and its interface to storage devices.

**[0010]** FIG. 3 is a block diagram illustrating one embodiment of a file system configured to detect identity-modifying operations on files.

10

**[0011]** FIG. 4A is a flow diagram illustrating one embodiment of a method of generating and storing records corresponding to identity-modifying file operations.

**[0012]** FIG. 4B is a flow diagram illustrating one embodiment of a method of  
15 importing records corresponding to identity-modifying file operations into a file mutation database.

**[0013]** FIG. 4C is a flow diagram illustrating one embodiment of a method of determining whether two files are in the same lineage pool.

20

**[0014]** FIG. 4D is a flow diagram illustrating one embodiment of a method of determining whether one file is an ancestor of another file.

**[0015]** FIG. 5 is a block diagram illustrating one embodiment of a file system  
25 configured to detect content access operations on files.

**[0016]** FIG. 6A is a flow diagram illustrating one embodiment of a method of generating and storing records corresponding to content access file operations.

[0017] FIG. 6B is a flow diagram illustrating one embodiment of a method of importing records corresponding to content access file operations into a file mutation database.

5

[0018] While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular  
10 form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

15

## DETAILED DESCRIPTION

### Storage system and file system overview

5 [0019] Turning now to FIG. 1, a block diagram of one embodiment of a storage system is shown. In the illustrated embodiment, storage system 10 includes a plurality of host devices 20a and 20b coupled to a plurality of storage devices 30a and 30b via a system interconnect 40. Further, host device 20b includes a system memory 25 in the illustrated embodiment. For simplicity of reference, elements referred to herein by a  
10 reference number followed by a letter may be referred to collectively by the reference number alone. For example, host devices 20a and 20b and storage devices 30a and 30b may be referred to collectively as host devices 20 and storage devices 30.

[0020] In various embodiments of storage system 10, host devices 20 may be  
15 configured to access data stored on one or more of storage devices 30. In one embodiment, storage system 10 may be implemented within a single computer system, for example as an integrated storage server. In such an embodiment, for example, host devices 20 may be individual processors, system memory 25 may be a cache memory such as a static RAM (SRAM), storage devices 30 may be mass storage devices such as  
20 hard disk drives or other writable or rewritable media, and system interconnect 40 may include a peripheral bus interconnect such as a Peripheral Component Interface (PCI) bus. In some such embodiments, system interconnect 40 may include several types of interconnect between host devices 20 and storage devices 30. For example, system interconnect 40 may include one or more processor buses (not shown) configured for  
25 coupling to host devices 20, one or more bus bridges (not shown) configured to couple the processor buses to one or more peripheral buses, and one or more storage device interfaces (not shown) configured to couple the peripheral buses to storage devices 30. Storage device interface types may in various embodiments include the Small Computer

System Interface (SCSI), AT Attachment Packet Interface (ATAPI), Firewire, and/or Universal Serial Bus (USB), for example, although numerous alternative embodiments including other interface types are possible and contemplated.

5     **[0021]**     In an embodiment of storage system 10 implemented within a single computer system, storage system 10 may be configured to provide most of the data storage requirements for one or more other computer systems (not shown), and may be configured to communicate with such other computer systems. In an alternative embodiment, storage system 10 may be configured as a distributed storage system, such  
10     as a storage area network (SAN), for example. In such an embodiment, for example, host devices 20 may be individual computer systems such as server systems, system memory 25 may be comprised of one or more types of dynamic RAM (DRAM), storage devices 30 may be standalone storage nodes each including one or more hard disk drives or other types of storage, and system interconnect 40 may be a communication network such as  
15     Ethernet or Fibre Channel. A distributed storage configuration of storage system 10 may facilitate scaling of storage system capacity as well as data bandwidth between host and storage devices.

20     **[0022]**     In still another embodiment, storage system 10 may be configured as a hybrid storage system, where some storage devices 30 are integrated within the same computer system as some host devices 20, while other storage devices 30 are configured as standalone devices coupled across a network to other host devices 20. In such a hybrid storage system, system interconnect 40 may encompass a variety of interconnect mechanisms, such as the peripheral bus and network interconnect described above.

25

**[0023]**     It is noted that although two host devices 20 and two storage devices 30 are illustrated in FIG. 1, it is contemplated that storage system 10 may have an arbitrary number of each of these types of devices in alternative embodiments. Also, in some

embodiments of storage system 10, more than one instance of system memory 25 may be employed, for example in other host devices 20 or storage devices 30. Further, in some embodiments, a given system memory 25 may reside externally to host devices 20 and storage devices 30 and may be coupled directly to a given host device 20 or storage  
5 device 30 or indirectly through system interconnect 40.

[0024] In many embodiments of storage system 10, one or more host devices 20 may be configured to execute program instructions and to reference data, thereby performing a computational function. In some embodiments, system memory 25 may be one  
10 embodiment of a computer-accessible medium configured to store such program instructions and data. However, in other embodiments, program instructions and/or data may be received, sent or stored upon different types of computer-accessible media. Generally speaking, a computer-accessible medium may include storage media or memory media such as magnetic or optical media, e.g., disk or CD-ROM included in  
15 storage system 10 as storage devices 30. A computer-accessible medium may also include volatile or non-volatile media such as RAM (e.g. SDRAM, DDR SDRAM, RDRAM, SRAM, etc.), ROM, etc, that may be included in some embodiments of storage system 10 as system memory 25. Further, a computer-accessible medium may include transmission media or signals such as electrical, electromagnetic, or digital signals,  
20 conveyed via a communication medium such as network and/or a wireless link, which may be included in some embodiments of storage system 10 as system interconnect 40.

[0025] In some embodiments, program instructions and data stored within a computer-accessible medium as described above may implement an operating system that may in  
25 turn provide an environment for execution of various application programs. For example, a given host device 20 may be configured to execute a version of the Microsoft Windows operating system, the Unix operating system, the Apple Macintosh operating system, or another suitable operating system. Additionally, a given host device may be configured

to execute application programs such as word processors, web browsers and/or servers, email clients and/or servers, and multimedia applications, among many other possible applications.

5    **[0026]**    During execution on a given host device 20, either the operating system or a given application may generate requests for data to be loaded from or stored to a given storage device 30. For example, code corresponding to portions of the operating system or an application itself may be stored on a given storage device 30, so in response to invocation of the desired operation system routine or application program, the  
10   corresponding code may be retrieved for execution. Similarly, operating system or application execution may produce data to be stored.

**[0027]**    Many operating system embodiments provide data and control structures for organizing the storage space provided by storage devices 30 into files. In various  
15   embodiments, the data structures may include one or more tables configured to store information such as, for example, the identity of each file, its location within storage devices 30 (e.g., a mapping to a particular physical location within a particular storage device), as well as other information about each file as described in greater detail below. Also, in various embodiments, the control structures may include executable routines for  
20   manipulating files, such as, for example, function calls for changing file identity and for modifying file content as described in greater detail below. Collectively, these data and control structures may be referred to herein as a file system, and the particular data formats and protocols implemented by a given file system may be referred to herein as the format of the file system.

25

**[0028]**    In some embodiments, a file system may be integrated into the operating system such that any access to data stored on storage devices 30 is governed by the control and data structures of the file system. Different operating systems may implement

different native file systems using different formats, but in some embodiments, a given operating system may include a file system that supports multiple different types of file system formats, including file system formats native to other operating systems. In such embodiments, the various file system formats supported by the file system may be referred to herein as local file systems. Additionally, in some embodiments, a file system may be implemented using multiple layers of functionality arranged in a hierarchy, as illustrated in FIG. 2.

[0029] FIG. 2 illustrates one embodiment of an operating system architecture and its interface to storage devices. In the illustrated embodiment, operating system 200 includes a user space 210 and a kernel space 220. User space 210 includes a plurality of processes 212A-C, each of which may correspond to a given user application. In some embodiments, some application processes 212 within user space 210 may be distinct from operating system 200. Such processes may be said to operate within an environment provided by operating system 200, or to operate “on top of” operating system 200. Each of processes 212 may be configured to access storage devices 230A-C through calls to application programming interface (API) 214. API 214 provides processes 212 with access to file system 205, which is configured to operate within kernel space 220. In one embodiment, storage devices 230 may be illustrative of storage devices 30 of FIG. 1. Also, in one embodiment, operating system 200, any of its components, and/or any of processes 212 may be configured to execute on one or more host devices 20 of FIG. 1, for example as program instructions and data stored within a computer-accessible medium such as system memory 25 of FIG. 1.

[0030] As described above with respect to storage system 10 of FIG. 1, a given host device 20 may reside in a different computer system from a given storage device 30, and may access that storage device via a network. Likewise, with respect to operating system 200, in one embodiment a given process such as process 212A may execute remotely and

may access storage devices 230 over a network. In the illustrated embodiment, file system 200 includes network protocols 225 to support access to the file system by remote processes. In some embodiments, network protocols 225 may include support for the Network File System (NFS) protocol or the Common Internet File System (CIFS) protocol, for example, although it is contemplated that any suitable network protocol may be employed, and that multiple such protocols may be supported in some embodiments.

[0031] File system 205 may be configured to support a plurality of local file systems. In the illustrated embodiment, file system 205 includes a VERITAS (VxFS) format local file system 240A, a fast file system (FFS) format local file system 240B, and a proprietary (X) format local file system 240X. However, it is contemplated that in other embodiments, any number or combination of local file system formats may be supported by file system 205. To provide a common interface to the various local file systems 240, file system 205 includes a virtual file system 222. In one embodiment, virtual file system 222 may be configured to translate file system operations originating from processes 212 to a format applicable to the particular local file system 240 targeted by each operation. Additionally, in the illustrated embodiment operating system 200 includes device drivers 224 through which local file systems 240 may access storage devices 230. Device drivers 224 may implement data transfer protocols specific to the types of interfaces employed by storage devices 230. For example, in one embodiment device drivers 224 may provide support for transferring data across SCSI and ATAPI interfaces, though in other embodiments device drivers 224 may support other types and combinations of interfaces.

[0032] In the illustrated embodiment, file system 205 also includes filter driver 221. In some embodiments, filter driver 221 may be configured to monitor each operation entering file system 205 and, subsequent to detecting particular types of operations, to cause additional operations to be performed or to alter the behavior of the detected operation. For example, in one embodiment filter driver 221 may be configured to

combine multiple write operations into a single write operation to improve file system performance. In another embodiment, filter driver 221 may be configured to compute a signature of a file subsequent to detecting a write to that file. In still another embodiment, filter driver 221 may be configured to store information, such as records,  
5 associated with particular files subsequent to detecting certain kinds of operations on those files, as described in greater detail below. It is contemplated that in some embodiments, filter driver 221 may be configured to implement one or more combinations of the aforementioned operations, including other filter operations not specifically mentioned.

10

**[0033]** It is noted that filter driver 221 is part of file system 205 and not an application or process within user space 210. Consequently, filter driver 221 may be configured to operate independent of applications and processes within the user space 210.

Alternatively, or in addition to the above, filter driver 221 may be configured to perform  
15 operations in response to requests received from applications or processes within the user space 210.

**[0034]** It is further noted that in some embodiments, kernel space 220 may include processes (not shown) that generate accesses to storage devices 230, similar to user space  
20 processes 212. In such embodiments, processes executing in kernel space 220 may be configured to access file system 205 through a kernel-mode API (not shown), in a manner similar to user space processes 212. Thus, in some embodiments, all accesses to storage devices 230 may be processed by file system 205, regardless of the type or space of the process originating the access operation.

25

**[0035]** Numerous alternative embodiments of operating system 200 and file system 205 are possible and contemplated. For example, file system 205 may support different numbers and formats of local file systems 240, or only a single local file system 240. In

some embodiments, network protocol 225 may be omitted or integrated into a portion of operating system 200 external to file system 205. Likewise, in some embodiments virtual file system 222 may be omitted or disabled, for example if only a single local file system 240 is in use. Additionally, in some embodiments filter driver 221 may be implemented  
5 within a different layer of file system 205. For example, in one embodiment, filter driver 221 may be integrated into virtual file system 222, while in another embodiment, an instance of filter driver 221 may be implemented in each of local file systems 240.

### **Tracking file identity change operations**

10

[0036] As described above, file system 205 may be configured to manage access to a plurality of files stored on storage devices 230. In many embodiments, each stored file may have an associated identity used by the file system to distinguish each file from other files. In one embodiment of file system 205, the identity of a file may be a file name,  
15 which may for example include a string of characters such as “filename.txt”. In embodiments of file system 205 that implement a file hierarchy, such as a hierarchy of folders or directories, all or part of the file hierarchy may be included in the file identity.

[0037] In the course of execution, operating system 200 and/or processes 212 may  
20 generate operations configured to modify the identity of one or more files managed by file system 205. In one embodiment, such identity-modifying operations may include any of the following: a file create operation, a file delete operation, a file rename operation, or a file copy operation. For example, a given process such as process 212A may receive a directive from a user to save work in a file with a corresponding identity that does not  
25 currently exist within file system 205, or to delete a specified file. Process 212A may then respectively generate a file create operation to create a file with the specified file identity, or a file delete operation to delete the specified file. Similarly, process 212A may receive a directive from a user to rename or copy a given file to a file with a

specified identity. Process 212A may then respectively generate a file rename operation or a file copy operation. In some embodiments, certain identity-modifying operations may be implemented using other identity-modifying operations. For example, a file rename operation may be implemented as a file create operation (specifying the identity  
5 of the target file of the rename) followed by a file delete operation (specifying the identity of the source file of the rename).

[0038] In one embodiment, file system 205 may be configured to detect various kinds of identity-modifying operations on files, and to store records of such operations. FIG. 3  
10 illustrates one such embodiment of a file system. The embodiment of file system 205 shown in FIG. 3 may include those elements illustrated in the embodiment of FIG. 2; however, for sake of clarity, some of these elements are not shown. In the illustrated embodiment, file system 205 includes filter driver 221, an arbitrary number of files 310a-n, and a respective named stream 320a-n associated with each of files 310a-n. File  
15 system 205 further includes a history stream 330, a file mutation database 340, and an update daemon 350. It is noted that a generic instance of one of files 310a-n or named streams 320a-n may be referred to respectively as a file 310 or a named stream 320, and that files 310a-n and named streams 320a-n may be referred to collectively as files 310 and named streams 320, respectively.

20

[0039] Files 310 may be representative of files managed by file system 205. Each of files 310 has a corresponding named stream 320. Each of named streams 320 may be configured to store information about its corresponding file, which may be referred to herein as metadata. In various embodiments, metadata may include information such as  
25 (but not limited to) the file identity, size, ownership, and file access permissions, as well as records corresponding to detected identity-modifying operations, as described below. It is noted that files 310 and named streams 320 may be physically stored on one or more storage devices, such as storage devices 230 of FIG. 2. However, for purposes of

illustration, files 310 and named streams 320 are shown as conceptually residing within file system 205.

#### Identity-modifying operation record generation and format

5

[0040] In one particular embodiment, file system 205 may be configured to detect an operation to modify the identity of a file 310, such as one of the identity-modifying operations described above. In such an embodiment, filter driver 221 may be configured to detect the identity-modifying operation when it is received by file system 205, or at  
10 some later time. Subsequent to detecting the identity-modifying operation, filter driver 221 may be configured to store a record of the detected operation in a named stream 320 corresponding to the target file of the operation. For example, if file 310a is the target of the detected operation, filter driver 221 may store a record of the operation in corresponding named stream 320a. It is contemplated that storage of a record may take  
15 place at any time subsequent to detection of the relevant operation. For example, in one embodiment, storage of the record may be delayed until the operation on file 310a is complete, while in another embodiment, storage of the record may occur prior to completion of the operation. In the latter case, if the operation is not guaranteed to complete (i.e., is speculative), filter driver 221 may provide a mechanism to delete a  
20 record stored in advance of its corresponding operation in case the operation does not complete.

[0041] The record stored by filter driver 221 subsequent to detecting an identity-modifying operation may in various embodiments include various kinds of information  
25 about the file 310 and the identity-modifying operation detected, such as the file identity, file type, file size, file owner, and/or file permissions, for example. In one embodiment, the record may include a file signature indicative of the content of file 310. A file signature may be a hash-type function of all or a portion of the file contents and may have

the property that minor differences in file content yield quantifiably distinct file signatures. For example, the file signature may employ the Message Digest 5 (MD5) algorithm, which may yield different signatures for files differing in content by as little as a single bit, although it is contemplated that any suitable signature-generating algorithm may be employed. In some embodiments, filter driver 221 may compute the file signature at the time the record of the identity-modifying operation is detected or stored, while in other embodiments filter driver 221 may use a file signature that was computed prior to detection of the operation.

10 [0042] In one embodiment, the record stored by filter driver 221 subsequent to detecting an identity-modifying operation may be generated and stored in Extensible Markup Language (XML) format, although it is contemplated that in other embodiments, any suitable format may be used. One example of an XML-format record is as follows:

```
15     <record sequence="1">
        <path>/test1/foo.pdf</path>
        <type>application/pdf</type>
        <user id=1598>username</user>
        <group id=119>groupname</group>
        <perm>rw-r--r--</perm>
20     <md5>d41d8cd98f00b204e9800998ecf8427e</md5>
        <size>0</size>
    </record>
```

Such a record may be appended to the named stream (for example, named stream 320a) associated with the file (for example, file 310a) having the file identity “/test1/foo.pdf” subsequent to a file create operation. In this case, the number associated with the “record sequence” field indicates that this record is the first record associated with file 310a. The “path” field includes the file identity, and the “type” field indicates the file type, which in one embodiment may be provided by the process issuing the file create operation, and in other embodiments may be determined from the extension of the file name or from header information within the file, for example. The “user id” field records both the numerical user id and the textual user name of the user associated with the process

issuing the file create operation, and the “group id” field records both the numerical group id and the textual group name of that user. The “perm” field records file permissions associated with file 310a in a format specific to the file system 205 and/or the operating system. The “md5” field records an MD5 signature corresponding to the file contents, and the “size” field records the length of file 310a in bytes. It is contemplated that in alternative embodiments, filter driver 221 may store records corresponding to identity-modifying operations that include more or fewer fields, as well as fields having different definitions and content.

10   **[0043]**   Filter driver 221 may be configured to append a record similar to the one illustrated above to the named stream 320 corresponding to a file 310 subsequent to detecting an identity-modifying operation of that file such as a create, delete, rename, or copy operation. Additionally, filter driver 221 may be configured to append a similar record to a named stream 320 corresponding to a file 310 when a process modifies the contents of file 310 without issuing an identity-modifying operation to the file. For example, in one embodiment filter driver 221 may be configured to detect a file close operation to a file 310 whose contents have been modified, where the file close operation is issued by the last process having the modified file open. In other words, multiple processes may have issued file open operations to a file 310 that is subsequently modified, and filter driver 221 may be configured to detect the last of such processes to issue a file close operation. Subsequent to detecting such a “last close” of the modified file 310, filter driver 221 may be configured to update the signature associated with the file 310 and to append a record including the updated signature to the named stream 320 corresponding to file 310. Filter driver 221 may thereby ensure that signatures reflected in records in named streams of files remain current without tracking each individual write of such files. Referring to the above example record, filter driver 221 may write the following example record to the named stream 320a of file 310a “/test1/foo.pdf” upon detecting the last close of the file following modification:

```
5      <record sequence="2">
        <path>/test1/foo.pdf</path>
        <type>application/pdf</type>
        <user id=1598>username</user>
        <group id=119>groupname</group>
        <perm>rw-r--r--</perm>
        <md5>b42455dadf928643d8df3171cca9216a</md5>
        <size>10597</size>
      </record>
```

10 As illustrated in this example record, the “md5” and “size” fields have been updated to reflect the modification to file 310a.

[0044] Certain identity-modifying file operations may involve more than one of files 310. For example, file rename and file copy operations may involve one or more source files and a destination file, where the destination file may or may not exist at the time the operation is performed. Subsequent to detecting identity-modifying operations involving more than one file, filter driver 221 may be configured to mark the existing records (if any) in the named stream 320 corresponding to the destination file 310 as “old,” and to append each record in the named streams corresponding to each source file to the named stream corresponding to the destination file. For example, subsequent to the modification of file 310a “/test1/foo.pdf” shown above, filter driver 221 may detect a file rename operation to rename file 310a “/test1/foo.pdf” to file 310b “/test1/destination.pdf”, which latter file may already exist and may have a number of records in its associated named stream 320b. Subsequently, filter driver 221 may mark the existing records associated with file 310b “/test1/destination.pdf” as old and associate each of the records associated with file 310a “/test1/foo.pdf” in its named stream 320a to the named stream 320b of file 310b “/test1/destination.pdf”, along with a new record indicating the identity change. Following this activity, the content of the named stream 320b of file 310b “/test1/destination.pdf” may include the following records:

```
30      <record sequence="1">
        <path>/test1/foo.pdf</path>
        ...
      </record>
```

```

    <record sequence="2">
        <path>/test1/foo.pdf</path>
        ...
    </record>
5    <record sequence="3">
        <path>/test1/destination.pdf</path>
        <type>application/pdf</type>
        <user id=1598>username</user>
        <group id=119>groupname</group>
10    <perm>rw-r--r--</perm>
        <md5>b42455dadf928643d8df3171cca9216a</md5>
        <size>10597</size>

        <oldrecord>
15            <record sequence="1">
                <path>/test1/destination.pdf</path>
                ...
            </record>
            ...
20    </oldrecord>
    </record>

```

where the first two records listed (with content omitted for clarity) may be identical to the first two records of file 310a “/test1/foo.pdf” as shown above, and the third record

25 indicates the change in file identity to “/test1/destination.pdf”. The other fields of the third record may be copied or linked from the most recent record of file 310a “/test1/foo.pdf” as indicated above. Further, records corresponding to file 310b “/test1/destination.pdf” prior to the identity change are shown being preserved (though their specific content is omitted for clarity) and delimited with the <oldrecord> indicator.

30 As shown above, the preserved old records are associated with a particular record (in this case, the third record), although in other embodiments it is contemplated that the old records may be associated with a different record or may constitute a standalone record separate from and not within the scope of another record.

[0045] It is noted that in some embodiments, following the aforementioned processing of records, filter driver 221 may be configured to delete the records associated with the source file 310 if the identity change operation is a file rename operation and to preserve the records associated with the source file 310 if the identity-modifying operation is a file copy operation. It is further noted that in some embodiments, file rename or copy operations may result in associated metadata records being duplicated in multiple named streams, whereas in other embodiments, metadata records may be associated with additional files by linking a pointer to an existing record into a named stream of a destination file rather than copying the record to the named stream of the destination file.

History stream and file mutation database

[0046] In the illustrated embodiment, file system 205 includes history stream 330. History stream 330 may be a named stream similar to named streams 320; however, rather than being associated with a particular file, history stream 330 may be associated directly with file system 205. In some embodiments, file system 205 may include only one history stream 330, while in other embodiments, more than one history stream 330 may be provided. For example, in one embodiment of file system 205 including a plurality of local file systems 240 as illustrated in FIG. 2, one history stream per local file system 240 may be provided.

[0047] In some embodiments, filter driver 221 may be configured to store a record in history stream 330 in response to storing a record corresponding to an identity-modifying operation in a given named stream 320. For example, in response to storing a record subsequent to detecting an operation to modify the identity or the content of a file 310 as described above, filter driver 221 may store a record indicative of the operation in history stream 330 as well as the identity of the file operated on. History stream 330 may thereby

provide a centralized history of the identity-modifying operations transpiring within file system 205.

[0048] In one embodiment, the record stored by filter driver 221 in history stream 330 may be generated in Extensible Markup Language (XML) format, although it is contemplated that in other embodiments, any suitable format may be used. Referring to the example above in which file 310a “/test1/foo.pdf” was created, modified, and then renamed to file 310b “/test1/destination.pdf”, in one embodiment history stream 330 may include the following example records subsequent to the rename operation:

```
10      <record>
          <op>create</op>
          <path>/test1/foo.pdf</path>
      </record>
      <record>
15          <op>modify</op>
          <path>/test1/foo.pdf</path>
      </record>
      <record>
20          <op>rename</op>
          <path>/test1/destination.pdf</path>
          <oldpath>/test1/foo.pdf</oldpath>
      </record>
```

In this example, the “op” field of each record indicates the operation performed, while the “path” field indicates the file identity of the file 310a operated on. In the case of the file rename operation, the “path” field indicates the file identity of the destination file 310b of the rename operation, and the “oldpath” field indicates the file identity of the source file 310a. It is contemplated that in alternative embodiments, filter driver 221 may store within history stream 330 records including more or fewer fields, as well as fields having different definitions and content.

[0049] Update daemon 350 may be configured as either a kernel-mode or a user-mode process operating within file system 205, although it is contemplated that in some

embodiments, update daemon 350 may be implemented external to file system 205. In the illustrated embodiment, update daemon 350 may scan the records stored in history stream 330 at regular or irregular intervals. If a valid record is found, for each destination file 310 recorded in the history stream (i.e., the file identified by the “path” field in the  
5 above example), update daemon 350 may be configured to access the corresponding named stream 320, and to convey the records stored therein to file mutation database 340. (In case a record stored in history stream 330 indicates that a given file 310 has been deleted, update daemon 350 may in one embodiment convey only that indication to file mutation database 340, as the named stream 320 corresponding to the deleted file 310  
10 may have also been deleted.) In one embodiment, update daemon 350 may convey all records stored in the corresponding named stream 320, while in other embodiments, update daemon 350 may convey only those records not previously conveyed to file mutation database 340. For example, in one embodiment each record in each named stream 320 may include a “scanned” field that may be tested and set by update daemon  
15 350, such that only unscanned records are conveyed to file mutation database 340. Similarly, in various embodiments update daemon 350 may mark records in history stream 330 as they are scanned, processing only unmarked records, or may delete them from history stream 330 after scanning.

20 [0050] It is noted that in an alternative embodiment, history stream 330 may be omitted from file system 205. In such an embodiment, update daemon 350 may be configured to scan all named streams 320 within file system 205 at regular or irregular intervals, conveying all or only modified records to file mutation database 340. Further, in another alternative embodiment, both history stream 330 and update daemon 350 may  
25 be omitted from file system 205. In such an embodiment, filter driver 221 may signal file mutation database 340 directly upon generating a record, such as via a software interrupt or function call, for example. Filter driver 221 may be configured to directly convey records to file mutation database 340, in which case records may not be stored within

named streams 320. Alternatively, file mutation database 340 may be configured to retrieve records directly from named streams 320 in response to receiving notification from filter driver 221 to do so.

5     **[0051]**     In the illustrated embodiment, file mutation database (FMD) 340 is a database integrated with file system 205, although it is contemplated that in other embodiments, FMD 340 may be implemented externally to file system 205. In various embodiments, FMD 340 may be configured as a kernel-mode or a user-mode process. FMD 340 may be configured to store records in the same format as the records stored in named streams 320  
10     and history stream 330, such as XML format records. However, it is contemplated that file mutation database 340 may implement any suitable database format or architecture. Further, in some embodiments, FMD 340 or update daemon 350 may be configured to convert records stored in one format within named streams 320 and history stream 330 to another format for storage within FMD 340. File system 205 may provide an API  
15     through which various processes may submit database queries to FMD 340, which may in turn be configured to respond to such queries.

20     **[0052]**     Numerous types of queries of FMD 340 are possible and contemplated, dependent on the type of information included in the records of identity-modifying operations generated by filter driver 221. For example, in one embodiment, whenever update daemon 350 conveys a record from a named stream 320 to FMD 340, FMD 340 may build a list identifying all files 310 having file signatures identical to the one included in the conveyed record. Subsequently, FMD 340 may be queried to identify all files sharing the file signature corresponding to a given file identity.

25

**[0053]**     Other types of queries may include queries to determine file lineage relationships among two or more files. Generally speaking, file lineage relationships refer to the relationships created among files as a result of identity-modifying operations.

Queries to determine file lineage relationships may include lineage pool queries and file ancestor queries, although other lineage relationships and associated queries are possible and contemplated. Files 310 may be considered to be members of the same lineage pool if they share a common file signature at some point in time, i.e., if each file has a record  
5 indicating the same file signature. A given file 310a may be considered to be an ancestor of a given file 310b if the first valid file signature of file 310b (i.e., the earliest record of file 310b including a file signature) matches some file signature of file 310a. Using such records and queries, file system 205 may be configured to detect and track the identities of files as those identities evolve through the execution of identity-modifying file  
10 operations. Such tracking may be useful, for example, in tracking the origins of properly or improperly modified files, or in implementing effective storage policies such as allowing files with identical content but different identities to share storage.

**[0054]** Other embodiments of file system 205 may be configured to determine file  
15 lineage relationships. For example, in one embodiment, FMD 340 may be omitted, and a query process may be configured to operate directly on records stored within named streams 320 to determine file lineage relationships. In another embodiment, named streams 320 may be omitted and records may be stored at the time of generation directly within FMD 340 or another type of repository for subsequent determination of file  
20 lineage relationships.

**[0055]** FIG. 4A and FIG. 4B illustrate embodiments of methods of generating and storing records corresponding to identity-modifying file operations and of importing such records into a file mutation database, respectively. Referring collectively to FIG. 1  
25 through FIG. 3 and FIG. 4A, operation begins in block 400 where an operation to modify the identity of a file is detected. In one embodiment, filter driver 221 of file system 205 may be configured to detect an identity-modifying operation as described above.

[0056] Subsequent to detection of the identity-modifying operation, a record of the operation is generated (block 402). In some embodiments, filter driver 221 may be configured to generate this record, and as described above, in some embodiments the record may be in the XML format and may include information about the operation, the  
5 file identity, a signature corresponding to the file, and other information as desired.

[0057] After the operation record is generated, it is stored in a named stream corresponding to the file (block 404). Additionally, a history record of the operation is stored in a history stream (block 406). As noted above, filter driver 221 may be  
10 configured in some embodiments to store the generated record in a named stream 320 corresponding to the file 310 targeted by the operation, and may additionally be configured to store a history record such as described above in history stream 330.

[0058] The method of FIG. 4B may in some embodiments operate in parallel to the  
15 method illustrated in FIG. 4A. For example, the method of FIG. 4B may be implemented within update daemon 350. Referring collectively to FIG. 1 through FIG. 3 and FIG. 4B, operation begins in block 410 where a history record corresponding to an identity-modifying operation is detected within the history stream. As described above, in one embodiment update daemon 350 may be configured to scan history stream 330 to detect  
20 history records not previously processed.

[0059] Once a history record is detected, the records stored in the named stream of the file indicated in the history record are accessed and conveyed to the file mutation database (block 412). As described above, in one embodiment update daemon 350 may  
25 be configured to access the named stream corresponding to a file indicated in the history record and convey the records included therein to file mutation database 340.

[0060] Other embodiments of these methods are possible and contemplated. For example, as noted above, in some embodiments of file system 205, history stream 330 may be omitted, and update daemon 350 may be configured to scan the entire file system to determine the presence of updated records. Also, in some embodiments update  
5 daemon 350 may be omitted and filter driver 221 may communicate directly with file mutation database 340.

[0061] FIG. 4C illustrates an embodiment of a method of determining whether two files are in the same lineage pool. Referring collectively to FIG. 1 through FIG. 3 and  
10 FIG. 4C, operation begins in block 420 where a request to determine whether two or more files are members of the same lineage pool is detected. For example, in one embodiment FMD 340 may be configured to detect a query corresponding to such a request.

[0062] Upon detecting such a request, the records corresponding to each file subject to  
15 the request may be examined (block 422). For example, in one embodiment FMD 340 may be configured to identify the database records corresponding to each subject file. In another embodiment, the records stored in named streams 320 corresponding to the subject files 310 may be scanned.

[0063] Subsequent to examination of the appropriate records, it may be determined  
20 whether the subject files share a common signature in any of their collective records (block 424). For example, in one embodiment FMD 340 may be configured to compare each unique signature indicated in the records of each subject file with each signature indicated in the records of every other subject file and to note signature matches. If a  
25 common signature exists among all subject files, the subject files may be determined to be members of the same lineage pool (block 426). Otherwise, the subject files may be determined to be members of different lineage pools (block 428).

[0064] FIG. 4D illustrates an embodiment of a method of determining whether one file is an ancestor of another file. Referring collectively to FIG. 1 through FIG. 3 and FIG. 4D, operation begins in block 430 where a request to determine whether a first file is an ancestor of a second file is detected. For example, in one embodiment FMD 340 may be  
5 configured to detect a query corresponding to such a request.

[0065] Upon detecting such a request, the records corresponding to each file subject to the request may be examined (block 432). For example, in one embodiment FMD 340 may be configured to identify the database records corresponding to each subject file. In  
10 another embodiment, the records stored in named streams 320 corresponding to the subject files 310 may be scanned.

[0066] Subsequent to examination of the appropriate records, it may be determined whether the first valid signature of the second file is included as a signature of the first  
15 file (block 434). For example, in one embodiment FMD 340 may be configured to compare the first valid signature of the second file with each unique signature indicated in the records of the first file and to note signature matches. If a matching signature exists, the first file may be determined to be an ancestor of the second file (block 436).  
Otherwise, it may be determined that the first file is not an ancestor of the second file  
20 (block 438). It is contemplated that in an alternative embodiment, the method of FIG. 4D may also be configured to determine whether the second file is an ancestor of the first file, for example by modifying the step at block 436 to include comparing the first valid signature of the first file with each unique signature indicated in the records of the second file and noting signature matches.

### **Tracking content access operations**

5 [0067] Referring once again to FIG. 2, as described above, file system 205 may be configured to manage access to a plurality of files stored on storage devices 230. In addition to each file having an associated file identity as described above, each file may have corresponding content. In various embodiments, such content may include data such as text data, image data, sound data, or application-specific data such as Microsoft Word data, for example. In other embodiments, file content may include executable  
10 code. For example, the content of a given file may include instructions that, when executed, perform the various functions of a program or application. File content may be stored via file system 205 on storage devices 230 using any encoding suitable for storage devices 230. For example, file content may be stored on storage devices 230 using a binary encoding.

15 [0068] In the course of execution, operating system 200 and/or processes 212 may generate input/output (I/O) operations configured to access the content of one or more files managed by file system 205. In some embodiments, such I/O operations may include a file read operation or a file write operation, and in one embodiment a file write  
20 operation may be further categorized as either an appending write operation (i.e., a write operation that appends content to a file) or a random write operation (i.e., a write operation that may overwrite the content of a file). For example, a given process such as process 212A may receive a directive from a user to open an existing file to read its content, or to save work in an existing file. Process 212A may then respectively generate  
25 a file read operation to read the content of the specified file, or a file write operation (such as an appending write operation) to modify the content of the specified file. In some embodiments, certain I/O operations may invoke or be invoked by some of the identify-modifying operations described above. For example, if a given file identity does not

exist, a file write operation to that file identity may result in a file create operation being performed, followed by a file write operation.

[0069] In some embodiments, file system 205 may be configured to aggregate file I/O operations on a per-process basis. For example, file system 205 may be configured to aggregate I/O operations on a given file performed by a particular process 212 from the time the file is opened until the time the file is closed by that particular process. This aggregation of I/O operations may be referred to herein as a content access operation. It is contemplated that the degree of aggregation of I/O operations into a single content access operation may vary in various embodiments. For example, in one embodiment all read and write I/O operations to a given file by a process 212 between the opening and closing of the given file may be aggregated into a single content access operation. In another embodiment, all such read I/O operations may be aggregated into one content access operation, and all such write I/O operations may be aggregated into a second content access operation. In yet another embodiment, each individual file I/O operation may correspond to a single content access operation.

[0070] In some embodiments, file system 205 may be configured to detect various kinds of content access operations on files, and to store records of such operations. FIG. 5 illustrates one such embodiment of a file system. The embodiment of file system 205 shown in FIG. 5 may include those elements illustrated in the embodiment of FIG. 2; however, for sake of clarity, some of these elements are not shown. Like the embodiment of FIG. 3, the embodiment of file system 205 illustrated in FIG. 5 includes filter driver 221, an arbitrary number of files 310a-n, and a respective named stream 320a-n associated with each of files 310a-n. File system 205 further includes a history stream 330, a file mutation database 340, and an update daemon 350. As above, a generic instance of one of files 310a-n or named streams 320a-n may be referred to respectively

as a file 310 or a named stream 320, and that files 310a-n and named streams 320a-n may be referred to collectively as files 310 and named streams 320, respectively.

5 [0071] Files 310 may be representative of files managed by file system 205. Each of files 310 has a corresponding named stream 320. Each of named streams 320 may be configured to store metadata about its corresponding file, as described above in conjunction with the description of FIG. 3. As described in greater detail below, in various embodiments, metadata may include records corresponding to detected content access operations, as well as the other kinds of information mentioned previously. As  
10 with the embodiment of FIG. 3, it is noted that files 310 and named streams 320 may be physically stored on one or more storage devices, such as storage devices 230 of FIG. 2. However, for purposes of illustration, files 310 and named streams 320 are shown as conceptually residing within file system 205.

15 Content access operation record generation and format

[0072] In one particular embodiment, file system 205 may be configured to detect an operation by a particular process 212 of FIG. 2 to access content of a file 310, such as one of the content access operations described above. In such an embodiment, filter driver  
20 221 may be configured to detect the content access operation when it is received by file system 205, or at some later time. Subsequent to detecting the content access operation, filter driver 221 may be configured to store a record of the detected operation in a named stream 320 corresponding to the target file of the operation. For example, if file 310a is the target of the detected operation, filter driver 221 may store a record of the operation in  
25 corresponding named stream 320a. It is contemplated that storage of a record may take place at any time subsequent to detection of the relevant operation. For example, in one embodiment, storage of the record may be delayed until the operation on file 310a is complete, while in another embodiment, storage of the record may occur prior to

completion of the operation. In the latter case, if the operation is not guaranteed to complete (i.e., is speculative), filter driver 221 may provide a mechanism to delete a record stored in advance of its corresponding operation in case the operation does not complete.

5

[0073] The record stored by filter driver 221 subsequent to detecting a content access operation may in various embodiments include various kinds of information about the file 310 and the content access operation detected, such as the file identity, file type, file size, file owner, file permissions, content access type, process identity, and/or process arguments, for example. In one embodiment, the record may include a file signature indicative of the content of file 310 as described in detail above, such as an MD5 signature, for example.

[0074] In one embodiment, the record stored by filter driver 221 subsequent to detecting a content access operation may be generated and stored in Extensible Markup Language (XML) format, although it is contemplated that in other embodiments, any suitable format may be used. One example of an XML-format record is as follows:

```

    <record sequence="4">
      <path>/test1/file.xls</path>
      <type>application/vnd.ms-excel</type>
      <user id="1598">username</user>
      <group id="119">groupname</group>
      <perm>rw-rw-r--x</perm>
      <md5>af662188a09d0b9998f710d744918bfe</md5>
      <size>15360</size>
      <date sec="1055278487">2003-06-10T20:54:47Z</date>
      <io>
        <write>append</write>
      </io>
      <process>
        <name>smbd</name>
        <args>/opt/VRTSsamba/bin/smbd -D
          -s/opt/VRTSsamba/lib/smb.conf</args>
        <pid>393</pid>
        <ppid>376</ppid>
        <pgrp>376</pgrp>
      </process>
    </record>
  </xml>
```

```
        </process>
    </record>
```

Such a record may be appended to the named stream (for example, named stream 320a) associated with the file (for example, file 310a) having the file identity “/test1/file.xls” subsequent to an appending write operation. In this case, the number associated with the “record sequence” field indicates that this record is the fourth record associated with file 310a. The “path” field includes the file identity, and the “type” field indicates the file type which in one embodiment may be provided by the process issuing the file create operation, and in other embodiments may be determined from the extension of the file name or from header information within the file, for example. The “user id” field records both the numerical user id and the textual user name of the user associated with the process issuing the file create operation, and the “group id” field records both the numerical group id and the textual group name of that user. The “perm” field records file permissions associated with file 310a in a format specific to the file system 205 and/or the operating system. The “md5” field records an MD5 signature corresponding to the file contents, and the “size” field records the length of file 310a in bytes.

[0075] Additionally, the “date” field records the date and time the record was created. The “io” field records information about the type of content access operation performed, and may include subfields specific to the operation type such as “read” and/or “write”; the “write” subfield may further delimit information regarding the type of write, such as “append” or “random.” The “process” field may include subfields recording information about the process performing the content access operation. The “name” subfield records the name of the process, and the “args” subfield records the arguments given when the process was invoked. The “pid,” “ppid,” and “pgrp” subfields record the process ID, the ID of the parent of the process, and the group ID of the process, respectively. It is contemplated that in alternative embodiments, filter driver 221 may store records corresponding to content access operations that include more or fewer fields, as well as fields having different definitions and content.

[0076] It is noted that in some embodiments, file system 205 may be configured to store records subsequent to detecting file content access operations, as just described, whereas in other embodiments, file system 205 may be configured to store records  
5 subsequent to detecting file identity-modifying operations as described above in conjunction with the description of FIG. 3. It is contemplated that in still other embodiments, file system 205 may be configured to store records corresponding to both content access operations and identity-modifying operations subsequent to detecting each respective type of operation. In one such embodiment, both types of records may be  
10 stored within a single named stream 320 corresponding to the file operated on, while in another such embodiment, each type of record may be stored in a distinct named stream corresponding to the file operated on. Further, in some embodiments storing both types of records, all stored records may follow the conventions described above for identity-modifying operations regardless of record type. For example, all stored records  
15 associated with a source file may be copied to the named stream of a destination file in the event of a file copy operation, and all stored records associated with a given file may be marked as “old” subsequent to that file changing identity, such as due to a file rename operation as described above.

20 History stream and file mutation database

[0077] In the illustrated embodiment, file system 205 includes history stream 330, which may be exemplary of history stream 330 of FIG. 3 and described in detail above. In some embodiments, filter driver 221 may be configured to store a record in history  
25 stream 330 in response to storing a record corresponding to a content access operation in a given named stream 320. For example, in response to storing a record subsequent to detecting an operation to access the content of a file 310 as described above, filter driver 221 may store a record indicative of the operation in history stream 330 as well as the

identity of the file operated on. History stream 330 may thereby provide a centralized history of the content access operations transpiring within file system 205.

[0078] As noted above, in one embodiment the record stored by filter driver 221 in history stream 330 may be generated in Extensible Markup Language (XML) format, although it is contemplated that in other embodiments, any suitable format may be used. Referring to the example above in which file 310a “/test1/file.xls” underwent an appending write operation, in one embodiment history stream 330 may include the following example record subsequent to the appending write operation:

```
10      <record>
           <op>append_write</op>
           <path>/test1/file.xls</path>
      </record>
```

15 In this example, as in the previous history record example, the “op” field of each record indicates the operation performed, while the “path” field indicates the file identity of the file 310a operated on. It is contemplated that in alternative embodiments, filter driver 221 may store within history stream 330 records including more or fewer fields, as well as fields having different definitions and content. For example, in one embodiment records corresponding to all types of write content access operations (e.g., appending and random) may be indicated simply as “modify” records within history stream 330 such as shown in the previous history record example.

[0079] Update daemon 350 may be exemplary of update daemon 350 of FIG. 3, described in detail above. As in that embodiment, update daemon 350 may be configured as a kernel-mode or user-mode process operating within file system 205 that may scan the records stored in history stream 330 at regular or irregular intervals. If a valid record is found, then for each destination file 310 recorded in the history stream (i.e., the file identified by the “path” field in the above example), update daemon 350 may be configured to access the corresponding named stream 320, and to convey the records

stored therein to file mutation database 340. As described in detail above, update daemon 350 may be configured to convey all records stored in the corresponding named stream 320, or only records newly created since the named stream 320 was last accessed by update daemon 350.

5

[0080] As in the embodiment of FIG. 3, it is noted that in an alternative embodiment, history stream 330 may be omitted from file system 205. In such an embodiment, update daemon 350 may be configured to scan all named streams 320 within file system 205 at regular or irregular intervals, conveying all or only modified records to file mutation  
10 database 340. Further, in another alternative embodiment, both history stream 330 and update daemon 350 may be omitted from file system 205. In such an embodiment, filter driver 221 may signal file mutation database 340 directly upon generating a record, such as via a software interrupt or function call, for example. Additionally, in some embodiments, filter driver 221 may be configured to store records subsequent to detected  
15 either identity-modifying operations or content access operations. As noted above, filter driver 221 may store both types of records in a single named stream 320 corresponding to a given file 310, or in separate named streams. Update daemon 350 may be appropriately configured to retrieve records from one or more named streams 320 according to each such embodiment.

20

[0081] In the illustrated embodiment, as for the embodiment illustrated in FIG. 3, file mutation database (FMD) 340 is a database integrated with file system 205, although it is contemplated that in other embodiments, FMD 340 may be implemented externally to file system 205. FMD 340 may be configured to store records in the same format as the  
25 records stored in named streams 320 and history stream 330, such as XML format records. However, it is contemplated that file mutation database 340 may implement any suitable database format or architecture. Further, in some embodiments, FMD 340 or update daemon 350 may be configured to convert records stored in one format within

named streams 320 and history stream 330 to another format for storage within FMD 340. File system 205 may provide an API through which various processes may submit database queries to FMD 340, which may in turn be configured to respond to such queries.

5

[0082] Numerous types of queries of FMD 340 are possible and contemplated, dependent on the type of information included in the records of content access operations generated by filter driver 221. Such queries may be configured to classify sets of files based on how content access operations indicate that such files are used. For example, in one embodiment, the class of log files (i.e., files used to log information regarding some aspect of a system's continuing operation) may exhibit a common set of characteristics, such as having appending writes but not random writes, having far fewer reads than writes, and having writes originating from a single process group rather than multiple process groups. In such an embodiment, a query may be designed and issued to FMD 10 340 to identify the files 310 having records of content access operations satisfying these characteristics. Based on this and similar classifications, different storage policies for file classes may be implemented. For example, in one embodiment file system 205 may assign files identified as log files through such a query to a lower-speed class of storage device, based on the heuristic that log files are typically infrequently accessed and 15 therefore relatively performance-insensitive. Numerous other queries corresponding to various file classes as well as storage policies for such file classes are possible and contemplated. 20

[0083] FIG. 6A and FIG. 6B illustrate embodiments of methods of generating and storing records corresponding to content access file operations and of importing such records into a file mutation database, respectively. Referring collectively to FIG. 1, FIG. 2, FIG. 5, and FIG. 6A, operation begins in block 600 where an operation to access 25

content of a file is detected. In one embodiment, filter driver 221 of file system 205 may be configured to detect an content access operation as described above.

5     **[0084]**     Subsequent to detection of the identity-modifying operation, a record of the operation is generated (block 602). In some embodiments, filter driver 221 may be configured to generate this record, and as described above, in some embodiments the record may be in the XML format and may include information about the operation, the file identity, a signature corresponding to the file, and other information as desired.

10   **[0085]**     After the operation record is generated, it is stored in a named stream corresponding to the file (block 604). Additionally, a history record of the operation is stored in a history stream (block 606). As noted above, filter driver 221 may be configured in some embodiments to store the generated record in a named stream 320 corresponding to the file 310 targeted by the operation, and may additionally be  
15   configured to store a history record such as described above in history stream 330.

**[0086]**     As in the case of identity-modifying operations described above, the method of FIG. 6B may in some embodiments operate in parallel to the method illustrated in FIG. 6A. For example, the method of FIG. 6B may be implemented within update daemon  
20   350. Referring collectively to FIG. 1, FIG. 2, FIG. 5, and FIG. 6B, operation begins in block 610 where a history record corresponding to a content access operation is detected within the history stream. As described above, in one embodiment update daemon 350 may be configured to scan history stream 330 to detect history records not previously  
25   processed.

**[0087]**     Once a history record is detected, the records stored in the named stream of the file indicated in the history record are accessed and conveyed to the file mutation database (block 612). As described above, in one embodiment update daemon 350 may

be configured to access the named stream corresponding to a file indicated in the history record and convey the records included therein to file mutation database 340.

5 [0088] Other embodiments of these methods are possible and contemplated. For example, as noted above, in some embodiments of file system 205, history stream 330 may be omitted, and update daemon 350 may be configured to scan the entire file system to determine the presence of updated records. Also, in some embodiments update daemon 350 may be omitted and filter driver 221 may communicate directly with file mutation database 340.

10

[0089] Although the embodiments above have been described in considerable detail, numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

15